CSE 435/535 Information Retrieval (Fall 2019)
Project Two: Boolean Query and Inverted Index

Due Date: October 12th 2019, 23:59 pm (EST)

Overview

In this project, you will be given a sample input text file consisting of Doc IDs and sentences. Based on this provided input text file, your task is to build your own inverted index using the information extracted from the given data. Your index should be stored as a Linked List in memory as the examples shown in textbook (Refer Chapter 1 – Boolean Retrieval). Having built this index, now you are required to implement a Document-at-a-time (DAAT) strategy to return Boolean query results. Finally, you are required to calculate a TF_IDF score to rank and sort the query results. Your implementation should be based on Python3 only.

Input Dataset

input_corpus.txt is a tab-delimited file where each line is a document; the first field is the document ID, and the second is a sentence. The two fields are separated by a tab.

Example:

```
1839	Feeling inspired? Make a meal for your family or roommates
1875	Make an effort to get to know someone you don't usually talk to.
2014	Help someone struggling with heavy bags
2095	Recycle 3 things today
2131	Help someone improve, give them constructive feedback;
```

Step 1: Build Your Own Inverted Index

As the examples in referred textbook, postings lists should be stored as **Linked Lists**.

You will need to construct your own index in which you will need to tokenize each sentence into terms by using a **white space tokenizer only. Do not remove any characters (such as . , - etc).**

Postings of each term should be ordered by **<u>increasing document IDs</u>**. For example:

term1 =	doc1	doc2	doc4
term2 =	doc2	doc4	doc5

Step 2: Boolean Query Processing

You are required to implement the following methods, and provide the results for a set of Boolean queries (AND/OR) **on your own index**. Results should be output as a .txt file in the required format.

1. <u>Get postings lists</u>

This method retrieves the postings lists for each of the given query terms. Input of this method will be a set of terms: term0, term1,..., termN. It should output the postings for each term in the following format:

GetPostings
term0
Postings list: 1000 2000 3000 ... (by increasing document IDs)
GetPostings
term1
Postings list: 1000 3000 6000 ... (by increasing document IDs)
...
GetPostings
termN
Postings list: 2000 7000 8000 ... (by increasing document IDs)

2. <u>Document-at-a-time AND query</u>

This function is to implement multi-term boolean AND query on the index using document-at-a-time (DAAT) strategy. Input of the function will be a set of query terms: term0, term1, …, termN. Output the following to the output file:

DaatAnd
term0 term1 ... termN
Results: 1000 2000 3000 ... (by increasing document IDs)
Number of documents in results: x
Number of comparisons: y

If the result of the query is empty then output:

DaatAnd
term0 term1 ... termN
Results: empty
Number of documents in results: 0
Number of comparisons: y

3. <u>**Document-at-a-time OR query**</u>

This function is used to implement multi-term boolean OR query on the index using DAAT. Input of this function will be a set of query terms: term0, term1, …, termN. Output the following to the output file:

DaatOr
term0 term1 ... termN
Results: 1000 2000 3000 ... (by increasing document IDs)
Number of documents in results: x
Number of comparisons: y

If the result of the query is empty then output:

DaatOr
term0 term1 ... termN
Results: empty
Number of documents in results: 0
Number of comparisons: y

NOTE: A comparison (for field "Number of comparisons") is counted whenever you compare two Document IDs during the union or intersection operation.

Step 3: TF-IDF Ranking

Extend your system from part 2 to perform simple TF-IDF scoring of the retrieved results. For this data set, there is no need to worry about any weight normalizations.

Use the following formulas to calculate TF-IDF:

TF(t) = (Number of times term t appears in a document) / (Total number of terms in the document).

IDF(t) = (Total number of documents / Number of documents with term t in it).

TF-IDF query

This function is used to rank the results from the DaatAnd and DaatOr using TF_IDF scoring. You need to return the *sorted* list of document IDs matching each of the queries from part 2.

TF-IDF
Results: 2000 1000 3000 … (document IDs by descending TF-IDF score)

If the result of the query is empty then output:

TF-IDF
Results: empty

NOTE:

There is no requirement for the names of your methods. However, you should submit a .py file **named exactly as "UBITName_project2.py"**, and your program should start running by executing the following command **on Timberlake**:

python3 UBITName_project2.py path_of_input_corpus path_of_output_result path_of_input_queries

Here, the first input parameter ***path_of_input_corpus*** should be able to accept a string indicating the path of the tab-delimited file containing the corpus. The second input parameter ***path_of_output_result*** should be able to accept a string indicating the path of the output file to write to, while the third, ***path_of_input_queries*** should be able to accept a string indicating the path to the input file containing the query terms.

The following assumptions can be made:

- The number of input query terms can be varied.
- All of the input query terms are selected from the vocabulary.
- Query terms should be processed in the order in which they are written in the query. Say, you should process term0 first, then term1, so on and so forth.
- DocumentID and corresponding sentence text will be separated by tab space in the input file reference by *path_of_input_corpus*.
- Input query terms will be separated by 'one blank space' in the *path_of_input_queries* file. You can assume there is no blank space within any query term.
- You will not have to handle cases with multiple Documents having the same **final** TF-IDF score.
- DO NOT use python build-in methods to do unions and intersections on postings lists directly. Create your own Pointers/References and have fun!
- Output should be formatted exactly the same as required. Otherwise you will not be able to get credits because grading will be automated!

Sample Input Output

The input file will be a .txt file containing multiple lines. Each line refers to a set of query terms which are separated by one blank space. For example:

```
term0 term1
term2 term3 term4 term5 term6 term7
term8 term9 term10
```

In this case, the output should be in the following order:

```
GetPostings term0 term1
DaatAnd term0 term1
TF-IDF term0 term1
DaatOr term0 term1
TF-IDF term0 term1
GetPostings term2 term3 term4 term5 term6 term7DaatAnd term2 term3 term4
term5 term6 term7
TF-IDF term2 term3 term4 term5 term6 term7
DaatOr term2 term3 term4 term5 term6 term7
TF-IDF term2 term3 term4 term5 term6 term7

GetPostings term8 term9 term10
DaatAnd term8 term9 term10
TF-IDF term8 term9 term10
DaatOr term8 term9 term10
TF-IDF term8 term9 term10
```

The corresponding output file for the input mentioned before will be:

```
GetPostings
term0
Postings list: 1000 2000 3000 …
GetPostings
term1
Postings list: 1000 2000 3000 …
DaatAnd
term0 term1
```

```
Results: 1000 2000 3000 …
Number of documents in results: x
Number of comparisons: y
TF-IDF
Results: 2000 1000 3000 …
DaatOr
term0 term1
Results: 1000 2000 3000 …
Number of documents in results: x
Number of comparisons: y
TF-IDF
Results: 2000 1000 3000 …

GetPostings
term2
Postings list: 1000 2000 3000 …
GetPostings
term3
Postings list: 1000 2000 3000 …
GetPostings
term4
Postings list: 1000 2000 3000 …
GetPostings
term5
Postings list: 1000 2000 3000 …
GetPostings
term6
Postings list: 1000 2000 3000 …
GetPostings
term7
Postings list: 1000 2000 3000 …
DaatAnd
term2 term3 term4 term5 term6 term7
Results: 1000 2000 3000 …
```

```
Number of documents in results: x
Number of comparisons: y
TF-IDF
Results: 2000 1000 3000 …
DaatOr
term2 term3 term4 term5 term6 term7
Results: 1000 2000 3000 …
Number of documents in results: x
Number of comparisons: y
TF-IDF
Results: 2000 1000 3000 …
GetPostings
term8
Postings list: 1000 2000 3000 …
GetPostings
term9
Postings list: 1000 2000 3000 …
GetPostings
term10
Postings list: 1000 2000 3000 …
DaatAnd
term8 term9 term10
Results: 1000 2000 3000 …
Number of documents in results: x
Number of comparisons: y
TF-IDF
Results: 2000 1000 3000 …
DaatOr
term8 term9 term10
Results: 1000 2000 3000 …
Number of documents in results: x
Number of comparisons: y
TF-IDF
```

Results: 2000 1000 3000 …

Grading and Evaluation

A successful implementation should be able to support all the functions mentioned before. It should also generate correct results in the required format. Failure in following the instructions and requirements will result in 0.

Rubrics:
Total points for this project: 10
Successfully rebuild index and correct output for GetPostings: 4
Correct results for DaatAnd / DaatOr: 1.5 + 1.5
Correct results for TF-IDF: 3

What to Submit

You should use cse-submit script to submit a .py file named exactly as "**UBITName_project2.py**". Double check and make sure your submission can be executed successfully **on Timberlake**. You need to use either "submit_cse435" (undergrad) or "submit_cse535" (grad), depending on your registration status. If you haven't used it, the instructions on how to use it is here: https://wiki.cse.buffalo.edu/services/content/submit-script

NOTE: Late submissions will NOT be accepted. The deadline is firm (i.e., October 12th 23:59 PM (EST)), if your timestamp is 12:00 AM, it is a late submission. Please start early.

FAQ's:

1. **How should I get started for this project?**

- First, make yourself familiar with fundamental concepts such as dictionary, postings, Inverted Index and Boolean operations. The best place to start is reading thoroughly the lecture slides and Chapter 1,2 of the referred textbook.

2. **What programming concepts are needed to complete this project?**
   - You are expected to know how to work with functions and be familiar with basic data structures such as Queue, Heap, HashMaps, Lists and so on.

3. **My program takes a while to execute. Would that be a problem in grading?**
   - Although the focus of this project is to test the correctness, we encourage you to be mindful of the data structure you are using for implementation. Unless it takes an unreasonably long time, you are fine in terms of grading but again please carefully analyze your code.